Parameterisable QR core

Parameterisable QR Core

Gaye Lightbody, Richard Walke*, Roger Woods, John McCanny
The Queen's University of Belfast, Ashby Building, Stranmillis Road,
Belfast BT9 5AH, Northern Ireland
* Defence, Evaluation and Research Establishment, Malvern , Worcs, WR14 3PS, England

Abstract

The design of a generic QR core for adaptive beamforming is presented. The work relies on an existing mapping technique that can be applied to a triangular QR array in such a way to allow the generation of a range of QR architectures. All scheduling of data inputs and retiming to include processor latency has been included within the generic representation.

1. Introduction

Adaptive filters play a key role in applications where the statistics of the incoming signals are unknown or changing. They are self-designing through the use of a recursive algorithm to calculate updates for the filter weights, such as the Recursive Least Squares (RLS) algorithm [1,2]. Although the RLS algorithm has a fast convergence rate it is considered highly computational, a factor that has hindered its use in real-time applications. A considerable body of work has been devoted to algorithms and VLSI architectures for RLS filtering with the aim to reduce this complexity. In particular, methods based on QR decomposition have been popular as they remove the computational bottleneck caused by the matrix inversion required to solve for the weights (e.g. using Givens rotation [3,4,5,6,7]). In addition, they may be implemented on a highly parallel triangular array processor [4,5], with the characteristics of a systolic array.

In this paper the design of a generic architecture is presented that offers a rapid implementation of the RLS algorithm solved by QR decomposition using the Squared Givens rotation (SGR) algorithm [6,7] - a derivative of the conventional Givens rotation algorithm [4,5]. This was chosen as it eliminated the square root operation from the boundary cell along with two of the four multiplications from the internal cells. However, even with this level of computation, the QR cells are complex and the number of processors within the QR array increases quadratically with the number of inputs. An N-input system would require $(N^2+N)/2$ QR processors. Typical applications such as adaptive beamforming require in excess of 40 inputs which would thus require a 20 chip solution [8]. Implementing the full QR array, i.e. a processor for each cell, would offer data rates far greater than those required by most applications. Consequently, the whole objective is to determine an efficient QR architecture which meets the desired performance criteria, in the minimum area and lowest power consumption possible. Therefore, an effective method of hardware reduction needs to be employed.

The focus of the work is primarily on the use of a novel hardware mapping used to develop a QR architecture which will meet the desired performance specification. Aspects of timing and scheduling are included in the analysis. The mapping used differs from a previous method, [9] in that both types of QR operation are performed on distinct processors. All processors are locally interconnected and used with 100% efficiency, thus retaining the systolic characteristics of the original triangular QR array. The QR operations were manipulated so that they formed a locally connected rectangular array, referred to as the processor array. Each column of operations were then assigned to an individual processor resulting in the linear architecture [7,8]. By partitioning this array in different ways, various levels of hardware reduction are achieved resulting in different sizes of linear or rectangular QR architectures (section 2). The effects of retiming and latency on the generic architecture is analysed in section 3 and a generic relationship is devised between the level of hardware reduction, latency and sample rate. The discussion is given in section 4.

2. Generic QR architecture

The signal flow graph (SFG) in figure 1 is a graphical representation of the SGR QR algorithm. It consists of two principle operations referred to as the boundary and internal cells. The QR decomposition transforms the input matrix X and vector y into an upper triangular

0-7803-5700-0/99/$10.00©1999 IEEE

matrix R and a vector u by a series of 2-dimensional Givens rotations, Q. The weights can then be solved by backsubstitution. The inputs enter the array at the top and are processed down through the cells on each clock cycle. Two rotation parameters, a and b, are calculated within a boundary cell to eliminate the input x_{BC} In the process, the R and u values are updated to account for the rotation. The values a and b, are then passed unchanged along the row of internal cells continuing the rotation. The output values of the internal cells, x_{OUT}, become the input values for the next row. The elements of the R matrix and u vector are stored within each of the processors until updated by the next rotation process. Meanwhile, new inputs are fed into the top of the array and the process repeats.

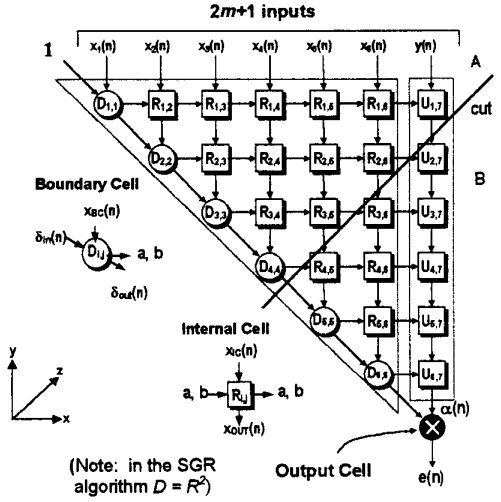


Figure 1: 7-input SGR QR array

Each cell within the SFG refers to a specific part of the QR operation (as denoted by the co-ordinate (i,j)). The cell position in the diagram shows the sequential relationship between parts of the algorithm and the position of the element of the R matrix and u vector held in the memory of each QR cell. It should be noted that the multiplier at the bottom of the array is considered as a boundary cell (labelled 7,7) for this explanation but would be implemented externally to the QR array.

Administering an efficient mapping procedure is complicated by the triangular shape and the position of the boundary cells along the diagonal. Consider the simple projection of operations of an N-input QR array from left to right onto a column of N processors. This leads to an architecture where the processors are required to perform both QR operations. Whilst the first processor is used 100% efficiently, this usage decreases down the column of processors such that the N^{th} processor is only used once every N cycles. This results in an overall efficiency of about 50%. Rader [9] solved the low processor usage, but in his solution, both QR operations still needed to be performed on the same type of processor. The mapping used in this paper enables the QR operations to be kept to distinct processors which are all locally interconnected and used 100% efficiently.

This mapping is achieved as follows. A cut is made in the triangular array after the $m+1^{th}$ boundary cell forming two triangles, A and B (figure 1). Triangle B is then mirrored in the x-axis and moved up along the y-axis resulting in a parallelogram shape shown in figure 2. Triangle B is moved to above A, forming the rectangular array shown in figure 3. A diagonal division is drawn so that equal numbers of cells are on either side (figure 3). The array is folded to interleave the processors placing the boundary cell operations down back on one diagonal. This results in the processor array (figure 4) from which a range of architectures with reduced number of processors can be obtained by dividing the array into partitions and then assigning each of the partitions to an individual processor.

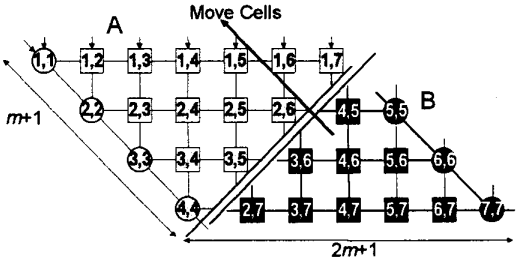


Figure 2: Modified array

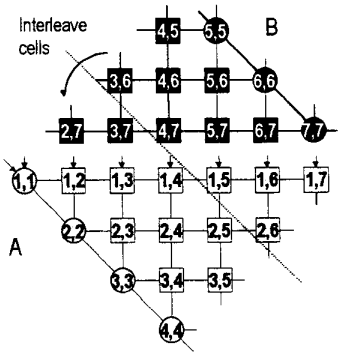


Figure 3: Folded array

By projecting down the diagonal it is possible to assign all the boundary cell operations to one boundary cell process and all the internal cell operations to a row of internal cell processors, resulting in the linear architecture shown in figure 6. Multiplexers present at the top of the array control the external system x inputs and the internal x values into the array. The rest control the different directions of data flow that occur between rows of the original array.

The processor array in figure 4 shows the sequence of operations to be performed on the linear architecture and represents the schedule. After 7 cycles of the linear architecture, a new QR update begins. This value is referred to as T_{QR} ,and is the number of cycles between the start of successive QR updates, which for the linear array was $2m+1$ cycles.

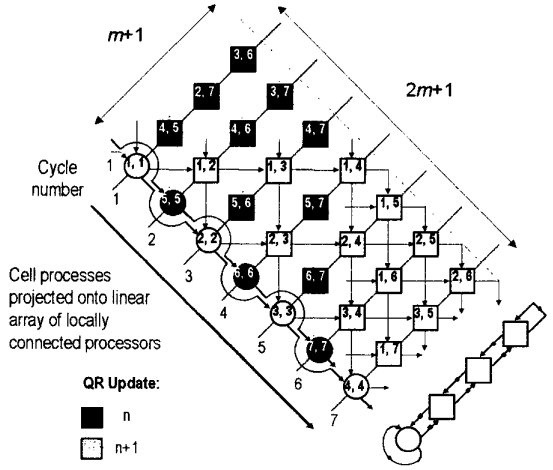


Figure 4: Processor Array

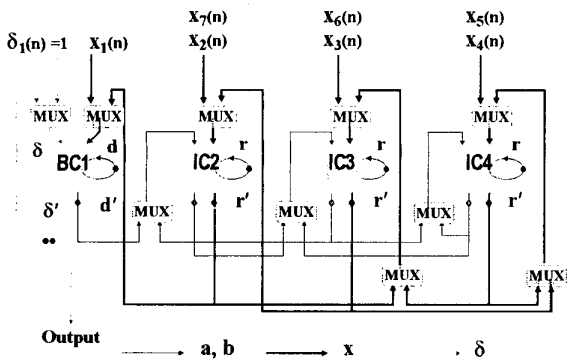


Figure 5: 4 cell linear architecture

The hardware level can be reduced further by assigning multiple columns of internal cell operations to each processor in a linear array. The example, (figure 6a) shows the three columns of operations assigned to one internal cell processor. The QR operations are scheduled onto the processors by using the schedule lines shown in the diagram. The sparse linear architecture is used 21 times to implement all the operations in the processor array, therefore, the next QR update starts 21 cycles after the previous update begins i.e. T_{QR} = 21. When multiple columns of internal cell operations are assigned to each processor (denoted as N_{IC}) then the number of iterations of the architecture is increased by this factor. Hence, for the sparse linear array, T_{QR} is expressed as the product of $(2m+1)$ and N_{IC}.

Alternatively, the operations could be performed on two linear arrays as depicted in figure 6(b). To balance the number of rows for each linear array, a dummy row of operations has been applied. On each clock cycle the rectangular array processor executes two rows of the original processor array therefore T_{QR} is 4. For the rectangular array, T_{QR} is determined by the number of rows of operations assigned to each row of processors, denoted as N_{rows}. The sparse rectangular array is a combination of the sparse linear and rectangular array. These derivations are summarised in table 1. The term, N_{IC}, is the number of columns of internal cell operations assigned to each processor, and N_{rows} is the number of rows of QR cell operations assigned to each line of processors

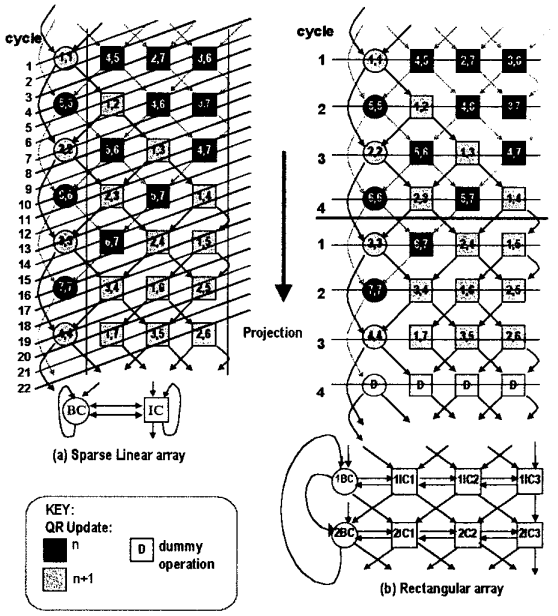


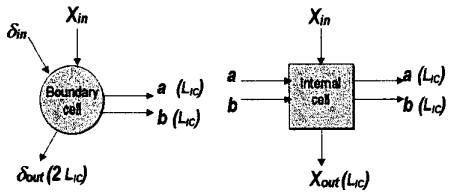
Figure 6: Example architecture mappings

Architecture	Size	T_{QR}
Processor array	(2m+1) rows ×(m+1) columns.	Latency in recursive loop
Linear:	1 BC and m ICs.	2m+1
Sparse linear:	1 BC, and $< m$ ICs	$N_{IC} \times (2m+1)$
Rectangular:	< (2m+1) rows, each with 1 BC and m ICs.	N_{rows}
Sparse rectangular:	<(2m+1) rows, each with 1 BC and $< m$ ICs.	$N_{rows} \times N_{IC}$
General	$(2m+1)/N_{rows}$ rows × m/N_{IC} columns	$N_{rows} \times N_{IC}$

Table 1: Summary of generic mapping applied to a QR array with $(2m+1)$ inputs

3. Retiming the generic architecture

The QR architectures so far have assumed that the QR cells have a latency of one clock cycle and the mapping is based on this factor; hence there will be no conflicts of the data inputs. However the inclusion of actual timing details within the QR cells may affect this guarantee of a valid data schedule. Embedding processor blocks with specific timing and pipelining information, coupled with the impact of truncation and internal word growth means that detailed retiming of the original SFGs of the QR cells must be performed before the architectures can be used to implement the QR architectures. The overall effect of retiming incurs latencies in the QR cells. For the purpose of this analysis it is not necessary to look at the retiming of the QR cells. Instead, generic latencies are assigned to the QR processors, and the scheduling analysis carried out accordingly, (figure 7). To maintain a regular data schedule, such as those shown in figures 4 and 6, the latencies of the QR cells need to be adjusted so that the x values and rotation parameters are output from the QR cells at the same time. The latency of the internal cell in producing these outputs can be expressed generically using a term L_{IC}. The latencies of the boundary cell in producing the rotation parameters, a and b, are also set to L_{IC} to keep outputs in synchronism. The latency of the boundary cell in producing the δ_{out} is set to double this value, i.e. $2L_{IC}$. This relates back to the original scheduling of the full QR array, whereby no two successive boundary cell operations are performed on successive cycles. By keeping the structure of the data schedule, so that the outputs of the QR cells appear in the same time instant, (except δ_{out}), then the retiming process comes down to a simple relationship, which shall be discussed next.



L_{IC} and $2L_{IC}$ are generic latencies for the QR cells, measured in clock cycles.

Figure 7: Generic latency of the QR cells

The latency has a major effect on the scheduling of the linear architecture. It stretches out the schedule of operations for each QR update, as shown in figure 8 for the linear array example, such that the n=2 iteration begins after $(2m+1)L_{IC}$ clock cycles. This is obviously not an optimum use of the architecture as it is only used every L_{IC}^{th} clock cycle, resulting in a low utilisation. New data samples need to be input at a much faster rate.

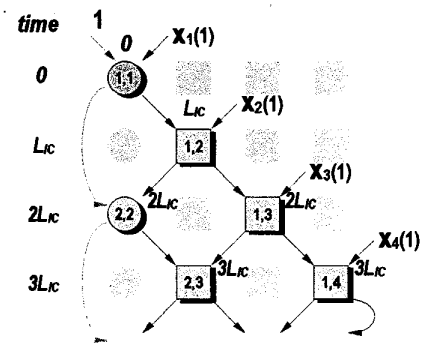


Figure 8: Stretched schedule

The factor, T_{QR}, is determined by the level of hardware reduction that is used to derive the architecture. It can be shown that a valid schedule which results in a 100% utilisation can be achieved by setting the latency to a value that is relatively prime to T_{QR}. That is, if the two values do not share a common factor other than 1 then their lowest common multiple will be their product. The time instance $T_{QR} \times L_{IC}$ does not represent a data collision. By choosing the value of T_{QR} to be equal to $2m+1$, then the QR operation that was on line to collide with a new QR operation will have just been completed. Figure 9 shows an example schedule for the 7-input linear array where L_{IC} is 3 and T_{QR} is 7. The shaded cells represent the QR operations from different updates that are interleaved with each other and fill the gaps left by the highlighted QR update. This diagram shows a 100% efficient schedule. The same relationship between L_{IC} and T_{QR} applies to the other QR architecture variants.

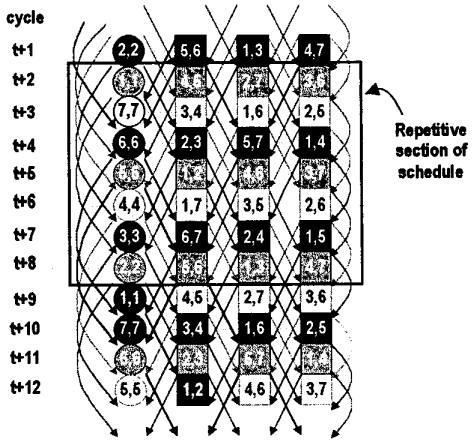


Figure 9: Schedule for a linear array with an internal cell latency of 3

A MATLAB program was written to mathematically model the time the inputs enter each cell of the linear array. Tests were then carried out for input collisions and processor usage over a range of values of L_{IC} and T_{QR}. Then a software model of the linear architecture built

using Cadence's SPW™ tool was updated to include latency. Simulations were performed to validate the retimed architecture proving that the linear architecture operates identically to the original triangular QR array.

4. Discussion

The details of a mapping has been presented which offers a means of generically deriving highly efficient architectures meet specific performance requirements. The architecture derivation can be applied to allow a range of resulting architectures with varied levels of hardware reduction. These architectures may be defined in four classes; linear, sparse linear, rectangular and sparse rectangular. Each type is a regular architecture with only local interconnections. Both the boundary and internal cell operations are performed on distinct processors. For the linear array the processors are used with 100% efficiency. For the sparse arrays the internal cell processors are used with 100% efficiency, while the usage of the boundary cell processor depends on the level of reduction in the sparse variant.

The derivation of the range of architectures begins with the formation of a processor array. This is a rectangular array of cells consisting of all the operations required by one QR update. It has the dimension, (2m+1) rows by (m+1) columns, where (2m+1) is the number of inputs of the original triangular array. The processor array is then used to derive the QR architectures.

The simple relationship between T_{QR} and L_{IC} is a key factor in achieving a high utilisation for these structures. The relationship gives a concise mathematical expression that is vitally important in the automatic generation of a generic QR architecture. This also addresses scheduling and retiming issues.

Table 2 shows example architectures derived from a QR array. The value for T_{QR} for the full QR array implementation is determined by the latency in the recursive loop of the QR cells (consisting of a floating point addition and a shift multiply function). For the example shown, the QR array needs to wait for 4 clock cycles for the calculation of the value in the recursive loop. Therefore this latency determines the sample rate of the system. This sample rate emphasises the poor return of performance of the full QR implementation at such a high cost of hardware.

A key outcome of this research is the elimination of the dependence of the latency within the recursive loops of the QR cells and the sample rate. If a QR array is to be implemented in full then the delay in the feedback loops will determine the sample rates and has therefore been a principal factor in the choice of QR algorithm. However, due to the level of hardware reduction in deriving the linear and rectangular QR architectures, the input sample rate is now controlled by the term T_{QR}. This value in turn is governed by the level and method of hardware reduction. Table 2 shows the performance achieved by implementing the full QR array can be met using about a quarter of the hardware.

5. References

[1] S. Haykin, *Adaptive Filter Theory*, Prentice Hall: Englewood Cliffs, NJ, 1986

[2] N. Kalouptsidis & S. Theodoridis, *Adaptive System Identification and signal processing*, Prentice Hall: Englewood Cliffs, NJ 1993.

[3] W. Givens, "Computation of plane unitary rotations transforming a general matrix to triangular form", *J. Soc. Ind. Appl. Math*, vol. 6, pp.26-50, 1958.

[4] W.M. Gentleman and H. T. Kung, "Matrix triangularisation by systolic array", *Proc. SPIE, (Real-Time Signal Processing IV)*, vol. 298, pp. 298-303, 1981

[5] J.G. McWhirter, "Recursive least squares minimisation using systolic array", *Proc. SPIE (Real-Time Signal Processing IV)*, vol. 431, pp. 105-112, 1983

[6] R. Döhler, "Squared Given's Rotations", *IMA J. of Numerical Analysis*, Vol. II, pp. 1-5, 1991

[7] R.Walke, *High Sample Rate Givens Rotations for Recursive Least Squares*, PhD Thesis, University of Warwick, 1997

[8] G. Lightbody, R. Walke, R. Woods, J. McCanny, "Rapid design of a single chip adaptive beamformer", IEEE Proc. on Signal Processing Systems, pp. 285-294, 1998.

[9] C.M. Rader "VLSI systolic arrays for adaptive nulling", *IEEE Signal Processing Magazine*, Vol. 13, No. 4, pp. 29-49, July 1996.

Architecture	Dimensions	Number of processors			T_{QR}	Data rate: Msamples/sec (Clock=100MHz)
		BC	IC	total		
Full QR array	Processor for each QR cell	45	990	**1035**	4	25
Rectangular 1	12 rows × 23 columns	12	264	**276**	4	25
Rectangular 2	3 rows × 23 columns	3	66	**69**	$(2m+1)/3=15$	6.67
Sparse-rectangular	3 rows × 12 columns (N_{IC}=2)	3	33	**36**	$2(2m+1)/3=30$	3.33
Linear	1 BC, 22 Ics	1	22	**23**	$2m+1=45$	2.22
Sparse-linear	1 BC, 11 ICs (N_{IC}=2)	1	11	**12**	$2(2m+1)=90$	1.11

Table 2: Example QR architectures with (2m+1=45) inputs